Abschlussprojekt

Abgabe: 12.08.18, 23:59

Implementiert ein Vier-Gewinnt-Spiel in C++. Benutzt werden darf C++-Standardbibliothek sowie ein Framework/eine Library zum automatisierten Testen/Mocken (z.B. Google-Test, QTest, …). Die Kommandozeile genügt als IO-Medium, eine GUI ist nicht erforderlich.

Das Spiel soll von zwei Spielern miteinander gespielt werden. Zwischen den Zügen der Spieler wird immer das aktuelle Spielfeld ausgegeben. Am Ende des Spiels wird der Gewinner ausgegeben. Alternativ hat man die Möglichkeit gegen vier unterschiedliche Arten von Bots zu spielen oder diese gegeneinander spielen zu lassen.

Der erste Bot agiert komplett zufällig – er wirft seinen Stein in eine zufällige Reihe. Bot Nr.2 legt seine Steine immer in die linkste Spalte, in der noch Platz ist. Bot Nr.3 legt seine Steine immer in die Spalte mit den wenigsten Steinen. Der vierte Bot ist ein schlau agierender Bot. Auf welche Art und weise er schlau spielt ist egal, er sollte aber in 85% der Fälle gegen alle anderen Bots gewinnen. Den Beweis dafür, dass er so häufig gewinnt ist mithilfe eines Tests abgedeckt, der die Bots jeweils 1.000 Mal gegeneinander antreten lässt. Dabei weiß er allerdings nicht, gegen welchen Bot er gerade spielt und kann es auch nicht herausfinden. Der schlaue Bot ist immer Spieler 2!

Abgabekriterien

- Das Projekt kompiliert ohne Fehler.
- Der Abgabetermin wird eingehalten.
- Das Programm ist selbstgeschrieben.
- Das Programm enthält in seinem Namen euren Namen und eure Matrikelnummer.
- Das Programm ist mithilfe von Git hochgeladen und abrufbar oder anderweitig als .zip hochgeladen. Mithilfe von Visual Studio oder Qt-Creator kann das Programm nach dem Klonen Out-Of-The-Box geöffnet und gestartet werden. Falls kleine Schritte zum Einrichten nötig sein sollten sind diese in einer ReadMe.txt des Hauptordners enthalten. Außerdem sollte in diesem Fall eine kompilierte Version des Spiels und der Tests vorliegen.
- Das Programm ist ein spielbares Vier Gewinnt, das korrekt den Sieg feststellt.

Bewertungskriterien

- Das Programm funktioniert korrekt (20)
- Das Programm stürzt auch bei unerwarteten Eingaben nicht ab (5)
- Bei Fehleingaben gibt das Programm korrekte Rückmeldungen (5)
- Ein Klassendiagramm beschreibt den Aufbau der Applikation (10)
    - Eine schriftliche Dokumentation kann (zumindest in Teilbereichen) dabei helfen, den Aufbau zu erklären, wird aber nicht erwartet
- Objektorientierte Prinzipien werden korrekt verwendet (15)
- Das Programm ist einfach lesbar (Sprechende Namen, simple Struktur, Dokumentation für *unklare* Programmteile) (10)
- DRY wird eingehalten (3)
- YAGNI wird eingehalten (3)
- SRP wird eingehalten (4)
- C++-Spezifika werden korrekt verwendet (10)
- Das Programm weist keine Memory-Leaks auf (10)
- Es gibt eine sinnvolle (Unit-)Testabdeckung (15)

    - Integration Tests werden lediglich für den Beweis der „Intelligenz“ des Bots erwartet

Tipps

- Legt den Großteil eures Programms direkt in einer Bibliothek an, sodass ihr beim Testen gegen diese linken könnt
    - Nutzt darin wenn möglich keine Konsolenein- oder ausgaben oder kapselt diese zumindest in einer eigenen Klasse
- Schreibt euch fürs Testen eine Methode oder Klasse, die euch aus einem String o.ä. ein Spielfeld erstellt/befüllt, das vereinfacht den Aufbau von Test-Cases
- Schreibt zunächst das Programm, den menschlichen Spieler, die trivialen Bots, sowie die Tests dafür und schreibt erst, wenn all das fertig ist den schlauen Bot
    - Schreibt dafür zunächst den Integrationstest
        - Dadurch bekommt ihr mit, wenn einer Bot schlau genug ist und lauft nicht in die Gefahr einen zu guten Bot zu programmieren, in den ihr zu viel Zeit steckt
        - Der schlaue Bot lässt sich sehr gut Test-Driven Entwickeln. Immer einen Test schreiben, der eine Bedingung prüft, danach diese Implementieren
            - Zwischendurch den Integrationstest laufen lassen
- Schreibt mir bei Fragen oder Problemen an meine private E-Mail-Adresse, ich bin im Urlaub
- Nutzt den mitgeschickten Mersenne-Twister als Zufallsfunktion, der ist bedeutend besser, als rand() aus der Standardbibliothek